MT-DIPS: a new data duplication integrity protection scheme for multi-tenants sharing storage in SaaS

Lin Li
School of Mathematics Science,
Shandong Normal University,
Jinan, China
Email: lilinsdu@126.com

Yongxin Zhang
School of Mathematics Science,
Shandong Normal University,
Jinan, China
and
Shandong Provincial Key Laboratory of
Software Engineering,
Jinan, China
Email: waterzyx@hotmail.com

Yanhui Ding*
School of Information Science and Engineering,
Shandong Normal University,
Jinan, China
Email: dingyanhui@gmail.com
*Corresponding author

Abstract: In SaaS, the data sharing storage mode and tenant isolation requirement present new challenge to traditional remote data duplication protection schemes. This paper aims at the new requirement of tenant data duplication protection in SaaS and presents a tuple sampling-based tenant duplication protection mechanism MT-DIPS (Duplication Integrity Protection Scheme for Multi-Tenants). Instead of data block sampling, MT-DIPS accommodates the data isolation requirement of different tenants by sampling tenants physical data tuples. Through periodical random sampling, MT-DIPS reduces the complexity on service provider side of verification object construction and eliminates the resource waste. Analysis and the experimental results show that if the damage rate of tenant data tuples is about 1%, the random sampling data number is about 5% of the total number of tuples. MT-DIPS makes use of homomorphism labels with auxiliary authentication structure to allow trusted third party verification without disclosing tenant data to relieve the verification burden on tenants’ client sides.

Keywords: SaaS; multi-tenant; duplication; integrity authentication; cloud computing.


Biographical notes: Lin Li is a Lecturer of Shandong Normal University. She received her Doctoral degree from Shandong University in 2014. Her research interests include cloud data manage and cloud data security.

Yongxin Zhang is a Lecturer of Shandong Normal University. He received his Doctoral degree from Shandong University in 2012 and his research interests include cloud data manage, cloud data integration and security.

Yanhui Ding is a Vice Professor and Master Supervisor of Computer Science and Engineering at Shandong Normal University. He received his doctor degree from Shandong University in 2010. His research interests include cloud data manage, cloud data integration and recommendation system, etc.

This paper is a revised and expanded version of a paper entitled ‘Multi-tenants Data Duplication Secure Storage in SaaS’ presented at the ‘IMIS 2015’, Santa Catarina, Brazil, 8–10 July 2015.
1 Introduction

Software as a Service (Aulbach et al., 2008), i.e. SaaS, is one important software delivery model in cloud computing and provides elastic extension, relatively inexpensive storage and computing resources for tenants. For many small and medium enterprises, SaaS is the best way to adopt advanced technologies. In SaaS, tenants subscribe to the software service through the web and do not care the implementation detail, while service providers take charge of software maintenance, management and upgrade. Single instance multi-tenancy is the common way adopted by the service providers, in which one instance can serve for multiple tenants. Meanwhile, in order to take full advantage of resources such as hardware and database, multiple tenants’ data is stored in one physical table such as universal table (Aulbach et al., 2009; Kong et al., 2010). In cloud, for data reliability, tenants always customise several data duplications and place them onto different data nodes of service provider.

Since tenants’ data is stored and processed at the platform of service providers, tenants’ ability of controlling their own data has been greatly weakened. The service provider may be untrustworthy and they may maliciously tamper, forge or delete tenant data duplication without tenant authorisation. For example, service providers may intentionally delete some long-term unused data within some data duplications of a tenant to save storage space for illegal interests, or tamper with some low rate used data to break the consistency maintenance update and spread the error data to all tenant duplications. How to prevent untrusted service providers from violating tenant data duplications integrity is an important issue that needs to be solved in SaaS.

Nowadays, for remote data duplication integrity protection, many researchers have focused on the problem of Provable Data Possession (PDP) and proposed different schemes to audit the data stored on remote servers (Wang et al., 2009; Stefanov et al., 2012; Nepal et al., 2011; Zhu et al., 2011; Wang et al., 2010; Liu et al., 2013). Those schemes did not fit well for the sharing storage mode of multi-tenants in SaaS because they needed to partition a whole file into many blocks, while in SaaS multiple tenants’ data is put into the same physical table, in which a single physical block may contain several tenants’ data. Directly applying existing scheme to multi-tenants sharing storage mode not only destroys the isolation requirement between tenants, but also increases the integrity verification cost because they need to distinguish the data owner for every blocks. So the data duplication integrity protection of SaaS needs to take into account the tenant data sharing storage characteristics along with the tenant data isolation requirements and system performance.

For the above problems, this paper puts forward a duplication integrity protection scheme for multi-tenants (MT-DIPS) based on the challenge-response model (Barsoum and Hasan, 2011). Different to the traditional approaches, in MT-DIPS, the sampled element is the tenants’ physical data tuples in universal table rather than the intersected data blocks of existing methods. Because the universal table can distinguish different tenant tuples by tenants’ unique identification’ (in this paper we call it TenantID). According to the sharing storage characteristic and tuples sample, MT-DIPS can accommodate the data isolation requirement of different tenants.

Second, in order to set up tuples, tuple-based sample sampling challenge-response model, MT-DIPS, constructs a new multi-tenants duplication authentication structure (MT-DAS). MT-DAS contains two layers, the lower is Tenant Layer (TL) and the upper is global index (GI). Tenant layer is a collection of merkle hash tree-based authentication structures of each data duplications and called duplication authentication trees (DAT). Global index is an index tree of the duplication authentication trees for all tenants duplication. MT-DAS can locate the authentication information of one tenants multiple duplication based on the GI quickly. Based on the two-layer structure, MT-DAS can separate different tenants’ data authentication information that is resident on the same data nodes to ensure the isolation requirement of different tenants. Besides, MT-DAS supports the data dynamic such as tuple modification, tuple deletion and tuple insertion with the help of DAT adjustment.

Third, to achieving the remote data authentication without the existence of local copies, MT-DIPS set up homomorphism label (Boneh et al., 2003) for each tuple of all duplication and combines them with MT-DAS together. Based on the homomorphic signatures, tenants can achieve the third party verification and do not have to preserve local copies for data verification to relieve the verification burden on tenant’s client side.

Finally, we give detailed security analysis along with the construction cost, storage cost, communication cost, authentication cost and tuples sampling cost.

The rest of this paper is organised as follows: in Section 2 we generally summarise the related work in data duplication secure storage in cloud. Section 3 puts forward the detailed definition of MT-DIPS and MT-DAS. Sections 4 and 5 give the initial construction, the integrity verification protocol and the adjustment strategy for dynamic data of MT-DIPS. Section 6 analyses the security and performance of MT-DIPS and experimental results. Section 7 summarises this paper.

2 Related work

For data secure storage in cloud, there are mainly two schemes, one is PDP (Ateniese et al., 2007; Ateniese et al., 2008; Erway et al., 2009; Hao et al., 2011) and the other is Proofs of Retrievability (POR) (Juels and Kaliski, 2007; Shacham and Waters, 2013; Bowers et al., 2009b). Among those PDP focused on how to judge whether the data on the remote node were damaged quickly while POR paid attention on identifying whether data had been damaged and how to recover the damaged data. Shuang et al. (2015) gave a classification and comparison of existing protocol of the provable data integrity.

Based on the above two schemes, for the multiple duplication integrity protection, Curtmola et al. (2008)
aimed at the data duplication protection and proposed MR-PDP scheme. In MR-PDP, users can periodically sample their data blocks through the challenge – response mode and use random mask code to encrypt different duplications to prevent service provider from deleting the whole data copy. Bowers et al. (2009a) introduced HAIL (High-Availability and Integrity Layer), which was a distributed cryptographic system that permits a set of servers to prove to a client that their stored file was intact and retrievable. But HAIL could not support the public verification and did not work well for the dynamic data modification. Juels and Kaliski (2007) put forward MF-RDC for some specific data update such as deletion and append on the duplication protection in cloud. MF-RDC mainly contained MR-PDP and MF-POR. MF-PDP model attached homomorphism tags to each data partition and combined those tags with block indexes to verify the remote data copy integrity. MF-POR encoded the files through error correction code, by this way files can be divided into multiple copies for storage to ensure the data retrievability. In MF-RDC, users didn’t need to download their data to local for the integrity verification, which reduced the communication cost and computational cost.

Besides, Halevi et al. (2011) proposed POW and POW can reduce the storage cost and network bandwidth of multiple copies of the same data in service provider. Based on POW, Zheng and Xu (2012) proposed POSD, POSD put PDP/POR scheme into POW and proved the security of POSD by CDH. Thakur et al. (2014) give a concrete implementation of cloud data integrity verification framework and algorithm implementation Barsoum and Hasan (2011) proposed two dynamic multi-copy integrity checking schemes TB-DMCPDP and MB-DMCPDP, those scheme could support multiple block dynamic operations such as modification, deletion, insertion and appendence.

The upper researches mainly based on the file data partition approaches that decomposed the file into a sequence data blocks and each block only contained one user’s data. They did not take the situation that one block may contain multiple users data. Putting those methods directly into multi-tenant application breaks the data isolation requirement of different tenants and increases the complexity of integrity verification.

For tenants’ structured data integrity protection in SaaS, Shi et al. (2011) focused on the case that service providers were not always trustworthy and promoted a metadata-driven data chunk-based secure data storage model for SaaS to ensure the data integrity. But it did not pay more attention to the situation that multiple tenants’ data stored in one same table crossed. Zhang et al. (2012) proposed a two-layer authentication tree, Double-Tree, for multi-tenants’ data copy protection based on MAB-Tree authenticating construction. Double-Tree could support data dynamic update and allows the third party to take periodic samples for the replicas data of users, but it did not consider the sharing storage model too. Meanwhile in Zhang et al. (2012) tenants needed to download remote data to local to verify which leaded to vast communication consumption waste. Li et al. (2014) presented a tenant-oriented data authentication scheme and provided data integrity assurance for multi-tenant sharing storage. But they did not resolve the duplication integrity problem.

At present most of the schemes are mainly aimed at the simple cloud data storage services, and few works pay attention to the data integrity issue of multi-tenants sharing storage in SaaS. So tenant data duplication integrity is a necessary issue that needs to be solved in SaaS.

3 Duplication integrity protection scheme for multi-tenants

3.1 Multi-tenants sharing storage model

This paper is mainly based on universal table storage model which is described in detail in Aulbach et al. (2009). The multi-tenants sharing storage model considered in our work is illustrated as Figure 1. In this model, tenants can log onto the service providers platform by internet web page and make use of their data with the leased application on the platform instead of downloading those data to native.

Suppose a tenant’s application logical view \( L = (A_1, A_2, \ldots, A_t) \), the physical view corresponding to the logical view \( L \) is \( LR = \{r_i\}_{i=1}^{s} \) and \( r_i = (GUID, AppID, TenantID, \{Ap_p, p = 1, ..., m\}) \). In \( r_i\), GUID is the global universal identification for each tuple, AppID is the application identification that tenants leased, TenantID is tenant’s unique identification. For reliability, tenant customises \( m \) data replicas \( \{Rj\}_{j=1}^{m} \) and decentralises those replicas to different data nodes of service provider.

3.2 Attack model

As shown in Figure 1, we assume the application platform (Application Mapping et al.) is trusted based on the existing research results of trusted platform (Brown and Chase, 2011; Alsouri et al., 2013). That means the inner data process is protected by the trusted platform and anyone (even the platform managers) could not damage the data on the application platform. Also we assume that all communications go through a security channel (Wang et al., 2015b; Wang et al., 2015c; Wang et al., 2015d) between tenants and service providers. By this assumption, we mainly pay attention to attack behaviours such as tamper, delete or forge tenant’s data on the physical storage of each data node from external and internal.

In SaaS, the internal attackers such as malicious data node managers may collude together to captivate tenants. For example, as in Figure 1, tenant C customises three duplications and puts them to data nodes 1, 2 and \( m \), the malicious data node managers may collaborate and only store one duplication in data node 2 to save the storage resource. When tenant C applies for data form data node 1 or \( m \), the inner attackers may transfer data from data node 2 to response the application. Tenant C could not distinguish the results coming from which data node because they look the same, while in fact the other two duplications may have been deleted. Compared with the external attackers, the internal attackers are more harmful and shady to tenant’s data.
3.3 Multi-tenants duplication authentication structure (MT-DAS)

MT-DAS contains two layers: the upper is global duplication authentication tree index (GI) and the lower is tenant layer (TL). Tenant layer covers multiple tenant duplication authentication trees (DAT), as shown in Figure 2.

3.3.1 Tenant layer

Tenant layer is a collection of merkle hash tree-based authentication structures for each data duplication, here we call them Duplication Authentication Trees (DAT). DAT is a transformation of merkle hash tree with fan-out \( f \geq 3 \). Each DAT corresponding to a duplication of tenant and DAT leaf node is the hash value of physical tuples in universal table. Figure 3 shows a simple schematic of DAT with fan-out \( f = 3 \).

**Definition:**

\[
DAT = \{\{DLnodei\}, \{DInodej\}, DRoot\}, i = 1 \ldots n, j = 1 \ldots m.
\]

- **DLnode** is the leaf node of DAT and \( DLnode_i = \{<k_l, h_l>\} \leq l \leq f \), \( k_l \) is the searching key of DAT, \( h_l \) corresponding to the tuples hash value with searching key \( k_l \). Here we suppose the domain of \( k_l \) is from 1 to \( N \).

- **DInode** means the inter node of DAT and \( DInode_i = \{<k_l, p_i, h_i>\} \leq l \leq f \), \( k_l \) is the searching key, \( p_i \) is the pointer that point to his child node that key \( k_i \) indicate, \( h_i \) is hash of the combination of all the child nodes hash value \( h_j = h(h_j \ldots h_i) \).

- **DRoot** is the root of DAT and it is signed by the tenant’s secret key.

3.3.2 Global index

Global Index (GI) is an index tree for DATs of all tenants. GI is similar to the B+ tree and contains three different nodes **GRoot**, **GINode** and **GLNode**:

- **GINode** is the collection of GI internal nodes and \( GINode = \{<K_i, p_i>\} \leq i \leq u \}, in which \( K_i \) is the searching key of GI, \( K_i = TenantID\&AppID \), TenantID is the unique identification of tenant and AppID is the remark of tenants application, \( p_i \) is the pointer that point to \( i \)th child node, \( u \) is the biggest fan-out of GI.

- **GLNode** is the collection of GI leaf nodes and \( GLNode = \{K, S < p_j, CN_j > \} \leq j \leq m \} \), \( S \) is the duplication signature and \( S = sig_k(h(IDR|hR)) \), in which IDR is the duplication label number of one tenant, IDR = \( h(TenantID\&AppID\&Rm) \) and it can uniquely mark the duplication information. \( h_R \) is the concatenation of all the \( \{R_j\} \leq j \leq m \) authentication tree root hash value \( h_R = h(CN_1|hR_1|\ldots|CN_m|hR_m) \), in which \( CN_j \) is duplication series number. \( p_j \) is the point that point to each DAT of tenant duplication.

- **GRoot** is the root node of the global duplication authentication tree index.

**Figure 2** Multi-tenants duplication authentication structure
VerifyProof, ExecUpdate, VerifyUpdate

collection tuples plaintext (Shannon et al., 1949). In this paper, tenant has a changed in the ciphertext, if there is a single bit change in the key, the output ciphertext tuples of the plaintext will be greatly with different secret key. According to the different encryption simply. To prevent this attack, we generate unique different inner attackers to cheat the tenants by storing only one data

From Section 3.2 we can see identical duplication enable the tenants (MT-DIPS) mainly contains seven algorithms:

- **KeyGen**(1\^t) \to (pk, sk): Tenant key generated, in this algorithm tenant random input a safe parameters 1\^t and get the key pair (pk, sk), pk is the public key, sk is the private key.

- **DATGen**(CNj, Rj) \to (CNj, DAT): Construct DATs for tenants m different duplication \{Rj\}\_\_m. This algorithm is accomplished by tenant on the application platform. For each Rj, tenant set-up corresponding DAT and put them together to the same data node.

- **TagGen**(sk, Rj) \to Ω: Generate authentication tags for each tuples in Rj. This algorithm is also done by tenant. The inputting is tuples and sk, and the outputting is authentication tags set Ω, which is corresponding to each tuples in Rj. In MT-DIPS, we use the BLS to generate homomorphic linear tags (Boneh et al., 2003).

- **GenProof**(Rj, Ω) \to P: For the sampled tuples, data node managers produce the verification proof P based on the native duplication Rj and authentication tag Ω to prove that they store tenants data honestly.

- **VerifyProof**(pk, P) \to \{True, False\}: Tenants or trusted third party verify the duplication integrity though the returned proof P and tenant public key pk, if success, output True, else False.

- **ExecUpdate**(Rj, \{Ω\}, Update) \to (\{R\_i\}, \{Ω\_i\}, P\_update): For the tenants updated data, data node managers receive the new data to generate the new authentication tags set Ω’, with update message P\_update and sent back P\_update, tenants.

- **VerifyUpdate**(pk, P\_update) \to (True, False): Tenants verify the correctness of returned update proof P\_update, if success, output True, else False.

The cryptography symbol that used in this paper includes:

- \pi\_m(\cdot) is a pseudo-random permutation(Wang et al., 2009): key × \{0;1\}^{log(\cdot)} \to \{0;1\}^{log(\cdot)}.

- \psi\_m(\cdot) is a pseudo-random function (Wang et al., 2009): key × \{0;1\}^* \to Zp.

- H(\cdot) is a map to point hash function (Barsoum and Hasan, 2011): \{0;1\}^* \to G.

- h is an cryptographic hash function, here we use SHA – 1 (Secure Hash Standard, 2010).

- Bilinear Map (Erway et al., 2009) is a map e: G × G \to G\_f with following properties:

  (i) Bilinear: \forall g\_1, g\_2 \in G and \forall a, b \in Zp there is: e(g\_1^a, g\_2^b) = e(g\_1, g\_2)^{ab};

  (ii) Computable: there exists an efficient algorithm for \forall g\_1, g\_2 \in G to compute e(g\_1, g\_2);

  (iii) Non-degenerate: e(g\_1, g\_2) \neq 1.

### 4 Integrity verification protocol of MT-DIPS

This section and the next section mainly discuss the detailed algorithm and protocol of MT-DIPS. From Section 3.4 we can see MT-DIPS contains seven basic algorithms and they have three stages in general: the initial construction, random sample verification and dynamic adjustment. In this section we give out the algorithm and protocol of the first two stages, the next section explains the dynamic update protocol in detail.

#### 4.1 The initial construction of MT-DIPS

The initial construction of MT-DIPS contains KeyGen, TagGen, and DATGen algorithms. Tenant runs KeyGen algorithm to get public key pair (a, γ), in which γ = g^a.
The TagGen algorithm mainly contains two stages: obfuscation duplication generation and authentication tag generation. In obfuscation duplication generation stage, tenant runs the data obfuscation algorithm to generate m different duplication \( \{R_j\}_{j\in\mathbb{N}} \). Different from the cloud storage data procession, we create obfuscation copies based on the physical tuples of storage data procession, we create obfuscation copies based on the physical tuples of \( L \) instead of the data blocks, so \( \{R_j\} = \{\tilde{Y}_j\}_{j\in\mathbb{N}} \). The obfuscation algorithm is similar to Curtmola et al. (2008) and we do not give the detail here.

In authentication tag generation stage, for each tuple in duplication \( \{R_j\} \), select random \( u \in G \) and compute \( \sigma_y = (H(\tilde{y}_j) \ldots u^{\tilde{y}_j})^\mu \). Authentication tag collection is \( \Omega_t = (\sigma_y)_{j\in\mathbb{N}} \). After the authentication tag generation, for each copy \( \{R_j\} \), tenant establishes verification DAT tree and put them along with authentication tag collection and corresponding duplication together to the same data node in cloud.

Based on the secret key, DATGen algorithm constructs DATs for tenants \( m \) different duplication \( \{R_j\}_{j\in\mathbb{N}} \). The DAT is commonly used to authenticate the correctness of the tuples. For dynamic update of tenant’s data, we need to authenticate both the tuples and their position in universal table to avoid attackers deleting the whole tuple. Also we need to make sure that a new inserted tuples has been putting to the request position. Therefore, to validate the completeness and correctness of tuples in duplication, the leaf nodes of DAT are treated in an ordered sequence by the searching key \( k \).

### 4.2 MT-DIPS Integrity Verification Process

In MT-DIPS, tenant can periodically check partial or all the data duplication on different data nodes. Here we mainly discuss the process of checking all the data duplication at one time. As shown in Figure 4, MT-DIPS integrity verification protocol contains five steps.

When tenants want to check their remote duplication, they first generate challenge token \( C \) and send \( C \) to service provider applications. On receiving the challenge token \( C \), application transmit \( C \) to each data node managers. According to \( C \), managers generate verify proof \( P \) base on the challenge tuples with corresponding DAT and authentication tags. Then application sends \( P \) back to tenants and tenants verify the correctness and completeness of the sampled tuples with proof \( P \).

#### 4.2.1 Challenge Token Generation

Tenant first specifics the tuple number \( c(1 \leq c \leq N) \) that to be sampled, then generates the pseudo-random permutation \( \pi_{\omega_i}(\cdot) \) key \( k_1 \) and pseudo-random function \( \psi_{\omega_i}(\cdot) \) key \( k_2 \) and sends the same algorithms \( \pi_{\omega_i}(\cdot) \) and \( \psi_{\omega_i}(\cdot) \) to service provider later.

With the key \( k_1 \) and \( k_2 \), tenant generates the challenge token \( C = \{ (i, v_i) \} \), \( C \) presents the tuple searching keys \( i \) and their corresponding random value \( v_i \) that to be checked, in which \( \{i\} = \pi_{\omega_i}(k_1)_{i\in\mathbb{N}}, i \in [1, N] \) and \( \{v_i\} = \psi_{\omega_i}(k_2)_{i\in\mathbb{N}} \). After the generation, the verifier sends \((c, k_1, k_2)\) to service provider.

#### 4.2.2 Verification Proof Generation

On receiving the \((c, k_1, k_2)\), data node manager runs GenProof algorithm to generate the response verification proof to prove that he has processed tenants data duplication honestly.

The verification proof \( P = (\sigma, \mu, \{H(\tilde{Y}_j), I_j\}_{(c,i)\in\mathbb{N} \times [1,N]}) \) in which:

(i) \( \sigma \) is the aggregation signature of all the sampled tuple authentication tags \( \sigma = \prod_{(c,i)\in\mathbb{C}} \sigma_i \) in which \( \sigma_i = \prod_{j=1}^{m} \sigma_{ij} \) is the aggregation signature of the same tuple corresponding to different duplication storage.

(ii) \( \mu \) is the authentication message that proves service provider stores tenants data duplication veraciously. \( \mu \) is the collection of all the duplication \( \mu_j = \sum_{v(c,i)\in\mathbb{C}} v_i \tilde{r}_i \).

(iii) \( I_j \) is auxiliary information in DATs that corresponds to the sampled tuples \( \{H(\tilde{Y}_j)\}_{(c,i)\in\mathbb{N} \times [1,N]} \). It contains all the brother nodes and their position message from leaf node to the root of the sampled tuples in the DAT.

#### 4.2.3 Integrity Verification

After receiving the verification proof \( P \) from service provider, tenant executes the VerifyProof algorithm to check the correctness and completeness of their remote data with \( P \), VerifyProof mainly contains:

(i) The verifier first combines \( \{H(\tilde{Y}_j), I_j\}_{(c,i)\in\mathbb{N} \times [1,N]}) \) with the tenant duplication number \( CN \) to reconstruct \( V = ID_p \).

(ii) Then the verifier compares \( e(S, g) = e(V, g') \) to check the correctness of \( P \). If the equation is false, the verification fails, else the verifier continue to check

\[
e(\sigma, g) = e\left(\prod_{(c,i)\in\mathbb{C}} \prod_{j=1}^{m} H(\tilde{r}_j)\cdot \sum_{v(c,i)\in\mathbb{C}} v_i\tilde{r}_i, g\right).
\]

If this equation is certificated right, the verification successes. The derivation process of this equation is:

\[
e(\sigma, g) = e\left(\prod_{(c,i)\in\mathbb{C}} \sigma_i, g\right) = e\left(\prod_{(c,i)\in\mathbb{C}} \prod_{j=1}^{m} \sigma_{ij}, g\right)
= e\left(\prod_{(c,i)\in\mathbb{C}} \prod_{j=1}^{m} H(\tilde{r}_j)\cdot \sum_{v(c,i)\in\mathbb{C}} v_i\tilde{r}_i, g\right)
= e\left(\prod_{(c,i)\in\mathbb{C}} H(\tilde{r}_j)\cdot \prod_{(c,i)\in\mathbb{C}} \prod_{j=1}^{m} H(\tilde{r}_j)\cdot \sum_{v(c,i)\in\mathbb{C}} v_i\tilde{r}_i, g\right)
= e\left(\prod_{(c,i)\in\mathbb{C}} H(\tilde{r}_j)\cdot \sum_{v(c,i)\in\mathbb{C}} v_i\tilde{r}_i, g\right)
= e\left(\prod_{(c,i)\in\mathbb{C}} H(\tilde{r}_j)\cdot \sum_{v(c,i)\in\mathbb{C}} v_i\tilde{r}_i, g\right)
\]
5 Adjustment strategy for dynamic data update

This section explains the dynamic update in MT-DIPS. In SaaS, the tenant data dynamic operation generally contains tuple modification (M), tuple insertion (I) and tuple deletion (D). For data modification, there is not need to adjust the tree structure and only to update the hash value on the path until root. For data deletion and data insertion, we insert or delete in the leaf node and then bottom-up change the path node until the root. Here we discuss separately in the following.

5.1 Tuple modification

Tuple modification is the most frequently used data operation in tenants’ application, here we mainly discuss the modification that does not apply on the searching key, the modification on searching key can be treated as delete the tuples. On receiving the update message, tenant generates new authentication tag $\sigma_i = \left( H\left( r_i' \right) \right)^{\alpha}$ and produces update message $update = \left( M, j, r_i', \left\{ \sigma_i \right\}_{1 \leq j \leq m} \right)$ indicate the data operation and $i$ is the searching key of the inserted tuples. On receiving the update message, data node managers process $ExecUpdate$ algorithm as following:

(i) Data node managers update the modified tuples $\left\{ \tilde{Y}_g' \right\}_{1 \leq j \leq m}$ in every $R$;

(ii) Update the authentication tags $\Omega_j^{'} \left( \sigma_1^{''}, ..., \sigma_j^{''}, ..., \sigma_n^{''} \right)$, $1 \leq j \leq m$, that equivalence to the modified tuples $\left\{ \tilde{Y}_g' \right\}_{1 \leq j \leq m}$;

(iii) Update the DATs of all duplications, replace the modified leaf nodes $h(H(\tilde{Y}_g))$ to new hash value $h(\tilde{Y}_g')$, compute the new root $h_g'$ and bottom to up and prepare the auxiliary information $\left\{ I_g \right\}_{1 \leq j \leq m}$ at the same time;

(iv) Service provider returns $P_{update} = \left( \left\{ I_g \right\}, \left\{ h(H(\tilde{Y}_g')) \right\}, S, h_g' \right)_{1 \leq j \leq m}$ to tenant.

When tenant receives the update proof, he first checks $e(S, g) = e(V, g')$ based on $\left\{ h(H(\tilde{Y}_g')) \right\}$ and $S$. If successful, tenant computes the new DAT roots from $P_{update}$, compares the result with $h_g'$ and returns true if they are consistent. Finally tenant generates new signature $S'$ and return $S'$ to service provider.

5.2 Tuple insertion

For data tuples insertion, we need to insert new DATleaf node and bottom-up change the path node until the root to adjust the whole DAT structure. The structure adjust is similar to Li et al. (2014) and we omit here.

Suppose tenant insert a new tuple $r^*$, all the duplication $R$ inserts $\left\{ r^* \right\}_{1 \leq j \leq m}$. According to the insert tuples, tenant generates new authentication tag $\sigma_i = \left( H\left( r_i' \right) \right)^{\alpha}$ and produces update message $update = \left( I, j, r_i', \left\{ \sigma_i \right\}_{1 \leq j \leq m} \right)$, $I$ indicates the data insertion operation and $i$ is the searching key of the inserted tuples. On receiving the update message, data node managers process $ExecUpdate$ algorithm as following:

(i) Update the inserted tuples $\left\{ \tilde{Y}_g' \right\}_{1 \leq j \leq m}$;

(ii) Update the authentication tags $\Omega_j^{'} \left( \sigma_1^{''}, ..., \sigma_j^{''}, ..., \sigma_{n_{+1}}^{''} \right)$, $1 \leq j \leq m$;

(iii) Update the DATs of all duplications, insert new leaf nodes $h(H(\tilde{Y}_g'))$, compute the new root $h_g'$ with new $h_g'$ bottom to up and prepare the auxiliary information $\left\{ I_g \right\}_{1 \leq j \leq m}$ at the same time;

(iv) Service provider returns $P_{update} = \left( \left\{ I_g \right\}, \left\{ h(H(\tilde{Y}_g')) \right\}, S, h_g' \right)_{1 \leq j \leq m}$ to tenant.

When tenant receives the update proof, he first checks $e(S, g) = e(V, g')$ based on $\left\{ H(\tilde{Y}_g') \right\}$ and $S$. If succeeds, tenant computes the new DAT roots from $P_{update}$, compares the result
with $h_x^*$ and returns true if they are consistent. Finally tenants generate new signature $S'$ and return $S'$ to service provider.

5.3 Tuple deletion

Tuple deletion ($D$) can be treated as the opposite process of tuple insertion. When tenant deletes a tuple, he sends update = ($D, t, null, null$) to service provider. On receiving the update message, data node managers process ExecUpdate algorithm as following:

(i) Delete tuples $Y_i$ in each $R$ and set up the new $R_d$;
(ii) Update the authentication tags $\Omega' = (\sigma_1, ... \sigma_{c-1}, \sigma_c, ... \sigma_n), 1 \leq j \leq m$;
(iii) Update the DATs of all duplication, delete leaf nodes $h(H(Y_i))$ and compute the new root $h_{f}''$ along with the auxiliary information $\{ij\} 1 \leq j \leq m$;
(iv) Service provider returns $P_{update} = (\{X_t\}, h(H(Y_i)), S, h_{f}''_{/i/j/m})$ to tenant.

The tenant process after receiving the update proof is similar with the above tuple modification and tuple insertion. After the verification, tenant generates new signature $S''$ and return $S''$ to service provider.

6 Analysis and experiment

6.1 Cost analysis

Suppose the physical tuple size is $|r|$, we analysis the construction cost, storage cost, communication cost and authentication cost in the following.

The construction cost of MT-DIPS mainly contains two parts, one is the authentication tag generation, the other is MT-DAS construction. Suppose $C_h$ is the cost of a single hash computing, $C_s$ is the cost of a single signature generation cost, $C_{hit}$ is the cost of map to point hash function cost, $C_i$ is the cost of verification of the tag. The initial construction cost is:

$$C = N \cdot m \cdot (C_h + C_{hit}) + m \cdot \left( \frac{N f - 1}{f - 1} \cdot C_s + C_i \right)$$

(1)

Suppose $|k|, |p|, |h|, |s|$ separately presents the size of searching key, pointer, hash value and authentication tag. The authentication structure size that stored on service provider is:

$$C = (N \cdot m + 1) \cdot |s|$$
$$+ m \cdot \left( \frac{N f - 1}{f - 1} \cdot (|k| + |p|) + \frac{N - 1}{f - 1} \cdot |p| \right)$$

(2)

Suppose the sampled tuple number is $c$ and key size of PRP, PRF is $|l|$. The size of challenge message is $|C| = 2 \ldots |l| + \log_2(c)$ and the size of returned proof $|P| = 2 \cdot |S| + c \cdot |log_{10}(N - 1) \cdot (f - 1) \cdot (|k| + |p| + |h|)|$. The generation cost of proof $P_i$ is:

$$C_{P_i} = C \cdot E_{x} + (c - 1) \cdot (m \cdot A_{c} + M_{c})$$
$$+ (m - 1) \cdot M_{c} + c \cdot m \cdot C_{hit}$$

(3)

Suppose MG presents multiplication cost in $G$, $E_{x}$ presents exponentiation cost in $G$, $M_{c}$ and $A_{c}$ presents multiplication and addition cost in $Z_p$. The verification cost of $P_i$ is:

$$C_{v} = \left( \frac{c - f - 1}{f - 1} + d \cdot (f - 1) \right) \cdot C_s$$
$$+ (c - 1) \cdot \left( N \cdot A_{c} + M_{c} \right)$$
$$+ c \cdot E_{x} + c \cdot m \cdot C_{hit} + (m - 1) \cdot M_{c} + 2C_{v}$$

(4)

6.2 Tuples sampling analysis

Suppose tenant physical view has $N$ tuples. If attacker damaged ($delete, tamper, forge$) $e$ tuples in one duplication, the failure probability is $\rho = \frac{e}{N}$. Assume $X$ is the discrete random variable of sampled tuples, so the probability $P_{d}$ that at least one damaged tuple to be detected is:

$$P = P\{X \geq 1\} = 1 - P\{X = 0\}$$
$$= 1 - C_{N-e}^{xN}$$

(5)

From equation 5, we can get $P \leq 1 - \left( \frac{N - e + 1 - e}{N - e} \right)^c$.

In MT-DIPS, the detection probability $P_{d}$ is an important parameter, it indicates at least how many tuples should be sampled after $e$ tuples were damaged. If $P$ is maintained constant, the relation between $c$ and $e$ is $e \leq (N - c + 1)(1 - \log_2(1 - P))$.

Figure 5 shows the relation between detection probability $P_{d}$ and sampled tuples $c$. The tuple number ranges from 5000 to 10,000 and random damage 100 tuples. We can see if we maintain $P_{d}$ as 99%, the random sample number $c$ is about...
500 from 10,000 tuples total. That means if the tenant contained in the logical view 10,000 data tuples and the damage rate is 1%, the random sampling data number is about 5% of the total number of tuples.

6.3 Security analysis

The security of MT-DIPS is based on CDH (Computational Diffie–Hellman) and discrete logarithm (Barsoum and Hasan, 2011). The CDH is given g, g, h ∈ Gp, for ∈ Zp it is difficult to compute h. The discrete logarithm problem is given g, h ∈ G, it is hard to get from h. Combining with the security analysis in Barsoum and Hasan (2011) and Wang et al. (2016, 2015a), here we prove that only the service providers storing tenants duplication honestly, they can produce the right verification proof to tenants.

Lemma: Assuming the hardness of CDH and discrete logarithm, the necessary condition for tenants admitting the returned proof P = {σ, μ} is service providers generate the μ = {μj}1≤j≤m from the original data duplication.

Proof: By the contradiction, we suppose the malicious attackers could forge the verification proof P′, P′ = {σ′, μ′} and {μ′j}1≤j≤m. On the other hands, suppose P = {σ, μ} is sent back by an honest service provider, in which σ = ∏i∈C σi, μj = ∑i∈C vi1j and μ = {μj}1≤j≤m.

For the correctness of MT-DIPS, P can satisfy e(σ, g) = e\left(\prod_{i \in C} \prod_{j=1}^{m} H(\hat{r}_j)^{\sum_{i \in C} \mu_i} \cdot u^\gamma\right).

Since σ′ ≠ σ, however, the assumption is σ′ passing the verification, so we can get e(σ′, g) = e\left(\prod_{i \in C} \prod_{j=1}^{m} H(\hat{r}_j)^{\sum_{i \in C} \nu_i} \cdot u^\gamma\right).

Define Δμj = μj′ − μ, as the upper assumption we get at least one Δμj(1 ≤ j ≤ m) is unequal to zero. Discomposing e(σ, g) and e(σ′, g), we get e(σ′, g) = e\left(u^{\sum_{i \in C} \nu_i} \cdot g^{\sum_{i \in C} \nu_i} \cdot u^\gamma\right).

So there is σ′ = σ′, as the upper assumption we get at least one Δμj(1 ≤ j ≤ m) is unequal to zero. Discomposing e(σ, g) and e(σ′, g), we get e(σ′, g) = e\left(u^{\sum_{i \in C} \nu_i} \cdot g^{\sum_{i \in C} \nu_i} \cdot u^\gamma\right).

So h = (σ′, σ′, γ, u^{\sum_{i \in C} \nu_i} \cdot g^{\sum_{i \in C} \nu_i})^{-1}. From the upper equation we can see if γ, ∑j=1m Δμj 0, h is the result of CDH problem which is conflict which lemma presupposition. So based on CDH, there must be σ′ = σ, in other words malicious attackers could not forge the verification proof at liberty. Meanwhile, we band tenant identification IDa into the verification proof to prevent service providers from replacing the signature of one tenant with other tenants.

In other way, if attackers forge μ′ = {μ′j}1≤j≤m and get P′ = {μ′, μ′}. There is:

e(σ, g) = e\left(\prod_{i \in C} \prod_{j=1}^{m} H(\hat{r}_j)^{\nu_i} \cdot u^{\gamma} \right),

So we can get: h = g^{\sum_{i \in C} \nu_i}.

We can see if x, ∑j=1m Δμj 0, h is the solution of discrete logarithm. It is conflicting with the assumption that at least one Δμj(1 ≤ j ≤ m) is unequal to zero, therefor μ′ = μ.

6.4 Experiment analysis

Our development environment is Eclipse-SDK-4.3.1-win 32 Bit on Windows 7 Professional Service Pack 1 with Intel core i3 2.3GHZ processor and 4GB RAM. The database server is Mysql5.1.22. We compare our work with Double-Tree based on simulation experiment at the authentication structure storage cost, communication cost and verification cost.

![Figure 6 Storage cost](image)

In authentication structure storage cost of Double-Tree and MT-DIPS, suppose tenants physical tuple size is 1 KB, we specify tenant tuple number ranging from 10 k to 50 k, and tenant customises three duplications. The storage cost of single duplication, all duplication, duplication with Double-Tree structure and duplication with MT-DIPS is shown in Figure 6. From Figure 6 we can see, since MT-DIPS attaches every tuple with their authentication tag, it has a higher storage cost than Double-Tree.
Then we fix tenant tuple number to $10k$ and sample tuple from 5% to 10% of the total number. Figure 7 shows the communication cost of Double-Tree and MT-DIPS. Figure 8 explains the verification cost. From the experiment result we find that MT-DIPS has a relative high cost of storage and verification compared with Double-Tree. Because we introduce the homomorphic tag into our authentication structure, those tags storage and verification need additional computing and storage resources. We can improve this problem though the sampling period; tenant can adjust the sampling frequency according to the peak distribution of service provider system to avoid the performance bottleneck.

Because of the homomorphic tag, MT-DIPS does not have to return the sampled tuples to tenant which greatly reduces the communication cost and native storage cost. Meanwhile MT-DIPS can authorise third parties for their data verification to release the burden by the homomorphic tag to avoid third parties from getting the privacy of tenants’ data. So from comprehensive consideration on privacy, communication and verification space, MT-DIPS is superior to Double-Tree.

7 Conclusion

This paper put forward a tenant-oriented duplication integrity checking scheme (MT-DIPS). MT-DIPS is a data integrity check scheme based on the challenge response model. MT-DIPS makes use of homomorphism labels with auxiliary authentication structure to allow trusted third party verification without disclosing tenant data. Through periodical random sampling, MT-DIPS reduces the complexity of service provider side verification object construction and eliminate the resource waste. There remain some problems of MT-DIPS for the future work such as how to improve tenant duplication obfuscation efficiency. And beyond those how to combine data integrity with data privacy in multi-tenant application is a challenging problem which remains later to solve.

Acknowledgements

This work is supported by National Natural Science Foundation of China No. 61303007, National Natural Science Foundation of China No. 61502218, Outstanding Young Scientists Foundation Grant of Shandong Province No. BS2014DX016, Research Fund for Excellent Young and Middle-aged Scientists of Shandong Province No. BS2013DX044, Outstanding Young Scientists Foundation Grant of Shandong Province under Grant No. BS2014DX016, PhD Programs Foundation of Ludong University under Grant No. LY2015033, Project of Shandong Province Higher Educational Science and Technology Program No. J15LN24.

References


